Domain Modeling and Grid Generation for Multi-block Structured Grids with Application to Aerodynamic and Hydrodynamic Configurations*

S.P. Spekreijse
J.W. Boerstoel
National Aerospace Laboratory NLR
P.O. Box 90502, 1006 BM Amsterdam, The Netherlands
phone 31-5274-8361, fax 31-5274-8210

P.L. Vitagliano
Alenia Aeronautica, Direzione Tecnica Trasporto/Ufficio di Aerodinamica
80038 Pomigliano d'Arco (Napoli), Italy

J.L. Kuyvenhoven
Fokker Aircraft B.V.
P.O. Box 7600, 1117 ZJ Schiphol-Oost, The Netherlands.

Abstract

About 5 years ago NLR , Alenia/Gat and Fokker started jointly the development of a flow simulation system for engine/airframe integration studies on propeller as well as jet aircraft. The initial system was based on the Euler equations and made operational for industrial aerodynamic design work. The system consists of three major components: – a domain modeller, for the graphical interactive subdivision of flow domains into an unstructured collection of blocks, – a grid generator, for the graphical interactive computation of structured grids in blocks, – a flow solver, for the computation of flows on multi-block grids. The industrial partners of the collaboration and NLR have demonstrated that the domain modeller, grid generator and flow solver can be applied to simulate Euler flows around complete aircraft, including propulsion system simulation. Extension to Navier-Stokes flows is in progress. Delft Hydraulics has shown that both the domain modeller and grid generator can also be applied successfully for hydrodynamic configurations. In this paper, an overview is given about the main aspects of both domain modelling and grid generation.

*This investigation was partly performed under contract 01604N with the Netherlands Agency for Aerospace Programs (NIVR).

1 Introduction

The underlying idea of the widely used and accepted multi-block approach is to subdivide a geometrical complex flow domain region into several smaller, more manageable regions, referred to as blocks. Typically there are several individual blocks in a given flow domain, each block having three computational coordinates. On each block there are six block-faces with two computational coordinates. On each face there are four face-edges, containing only one computational coordinate. Furthermore, each edge has two vertices.

A grid in a block is represented by a number of discrete grid points, ordered in a three dimensional array.

In general, the flow domain may be subdivided into any conceivable structure provided that cell to cell matching on block boundaries is maintained. This does not require that one block-face of a given block match exactly with a block-face of another block, only that each cell on an interior block-face match with a cell of another interior block-face. In our approach, it is possible to subdivide any block-face into two subfaces. A face which consists of two subfaces is called a compound face. Each subface of a compound face is also allowed to be compound again. A face which is not compound is called elementary. Thus it is possible to subdivide a block-face into any number of elementary subfaces. With the concept of compound faces, two individual blocks may be connected to each other by a face which is a subface of one of the six block-faces of both blocks. This feature is called " partial block boundary interfacing", i.e. any part of one block-face may be connected to any compatible part of another block-face [1],[2]. In most applications this flexibility allows the user to define much fewer and larger blocks than would be the case with "complete boundary interfacing". An illustration of this effect is given by a simple 2D example (Fig. 1).

The use of compound faces also requires the use of compound edges. A compound edge is an edge which consists of two subedges. Each subedge is also allowed to be compound again. An edge which is not compound is called elementary. Thus it is possible to subdivide a face-edge into any number of elementary subedges. It is clear that a compound face must have at least two opposite compound face-edges.

A multi-block system requires domain decomposition, i.e. the subdivision of the flow domain into suitable blocks. In our approach, domain decomposition is done by a so called domain modeller which is a graphical interactive code operational on Silicon Graphics workstations. During domain decomposition, a user may interactively create vertices, edges, faces and blocks. Tools are available to create edges and faces with a user-defined geometrical shape (such as interior block-interfaces and block-edges), and edges and faces which are constrained to a certain surface shape (particularly those conforming to the geometry).

Output of the domain modeller are a topology and geometry file. The topology file describes the arrangement of the blocks and the geometry file describes the geometrical position of the vertices and the geometrical shape of edges and faces. The topology and geometry files are the main input files of the grid generator which is also a graphical interactive code operational on Silicon Graphics workstations. Output of the grid generator

is a grid file which contains the grid points in the blocks and the topological information about the arrangement of the blocks. The grid file is the main input file for a flow solver.

In this paper, an overview is given about the main aspects of domain decomposition and grid generation. The usability of the applied domain decomposition and grid generation techniques is demonstrated for both aerodynamic and hydrodynamic configurations. A general overview of the complete system (including flow simulation) may be found in [3].

2 Domain decomposition

Computer aided design (CAD) has become a standard tool in the aerospace industry for geometrically defining developing configurations, and the majority of advanced designs which are used in CFD studies have indeed been created on CAD systems. The differences in the various CAD systems, however, have made it necessary to establish one universal format for direct use during interactive domain modelling. Therefore, an entire configuration surface is represented by a set of input configuration surfaces, each surface containing a two dimensional ordered array of physical points. The input configuration surfaces are input for the domain modeller.

The goal of domain modelling is the creation of a topology and geometry file, which are required input files for the grid generator. The topology file defines the topology of a multi-block system and is described in subsection 2.1. The geometry file defines the geometrical position of the vertices, and the geometrical shape of the edges and faces, and is described in subsection 2.2. The creation of the topology and geometry file during a graphical interactive domain decomposition is described in subsection 2.3.

2.1 Topology definition

The blocks,faces,edges and vertices in a particular multi-block system are represented as $\{B\},\{F\},\{E\},\{V\}$ which are so called " label sets ". The label sets are subsets of $\mathcal{N}$: the set of positive natural numbers.

The topology of a multi-block system describes the arrangement of blocks. The arrangement of the blocks is defined by connectivity relations between the label sets. Only five connectivity relations define the complete arrangement of the blocks in a particular multi-block system.

The first one relates each block B to an *ordered* set of six block-faces:

$$\forall\, B \in \{B\}:\ B \mapsto (F_1, F_2, F_3, F_4, F_5, F_6). \tag{1}$$

The second one relates each face F to an *ordered* set of four face-edges:

$$\forall\, F \in \{F\}:\ F \mapsto (E_1, E_2, E_3, E_4). \tag{2}$$

The third one relates each edge E to an *ordered* set of two edge-vertices:

$$\forall\, E \in \{E\}:\ E \mapsto (V_1, V_2). \tag{3}$$

Compound faces and compound edges are introduced in order to allow partial block boundary interfacing. A compound face consists of two subfaces. Each subface of a compound face is also allowed to be compound again. A face which is not compound is elementary. The set of compound and elementary faces are denoted as $\{F^c\}$ and $\{F^e\}$. Thus $\{F\} = \{F^e\} \cup \{F^c\}$. The fourth connectivity relation

$$\forall\, F \in \{F^c\} :\ F \mapsto (F_1, F_2). \tag{4}$$

gives the two subfaces of each compound face.

Similarly, a compound edge consists of two subedges. The set of compound and elementary edges are denoted as $\{E^c\}$ and $\{E^e\}$. Thus $\{E\} = \{E^e\} \cup \{E^c\}$, and the mapping

$$\forall\, E \in \{E^c\} :\ E \mapsto (E_1, E_2). \tag{5}$$

gives the two subedges of each compound edge.

For a particular multi-block system, the topology file, which is an output file of the domain decomposer and an input file of the grid generator, contains nothing else as the definition of these five connectivity relations.

The five connectivity relations should obey certain rules such that the topology is meaningful. For instance, a block should contain exactly twelve edges and eight vertices, and a face should contain exactly four vertices. The rules are such that the block-edges, block-vertices and face-vertices are uniquely defined.

The ordering in the relations (1),(2) and (3) are also used to define the orientation of all blocks, faces and edges in a particular multi-block system. In a block, the first coordinate runs from face F_1 to face F_2, the second coordinate runs from face F_3 to face F_4, the third coordinate runs from face F_5 to face F_6. Thus the three pairs of opposite faces in a block are (F_1, F_2),(F_3, F_4) and (F_5, F_6). In a face, the first coordinate runs from edge E_1 to edge E_2, the second coordinate runs from edge E_3 to edge E_4. Thus (E_1, E_2) and (E_3, E_4) are the two pairs of opposite edges in a face. Finally, the coordinate direction of an edge is from vertex V_1 to vertex V_2. The ordering in relations (4) and (5) is of no importance.

The relation (1),(2) and (3) are also used to identify all kinds of geometrical degenerations. For instance, an edge E with two equal vertex labels, i.e. $E \mapsto (V_1, V_2)$ and $V_1 = V_2$, is an edge whose curve-shape is degenerated to a point. This is consistent because the orientation of an edge with two equal vertex labels is undefined. A face with two equal opposite edges, i.e. $F \mapsto (E_1, E_2, E_3, E_4)$ and $E_1 = E_2$ or $E_3 = E_4$, is face with a surface shape degenerated to a curve. If both pairs of opposite edges are equal then the surface shape is a point.

2.2 Geometry definition

The geometrical shape of a vertex is a physical point. The geometrical position of all vertices in a particular multi-block system are stored on the geometry file.

The geometrical shape of an elementary edge is a curve. A default elementary edge is an edge of which the edge-curve shape is a straight line segment between the two edge-vertices.

A non-default elementary edge is an edge of which the edge-curve shape is described by an ordered one dimensional array of physical (control) points. The ordering is defined by the topology relation (3): the first control point is close to the position of vertex V_1, the last control point is close to the position of vertex V_2. Cubic Hermite interpolation between the control points is used to describe the curve continuously. A correction procedure is automatically performed such that the curve matches the two edge-vertices exactly. The control points of the non-default elementary edges in a particular multi-block system are also stored on the geometry file.

The geometrical shape of an elementary face is a surface. A default elementary face is a face of which the face-surface shape is defined by the geometrical curve shape of the four face-edges only (by bilinear transfinite interpolation in which arc length scaled coordinates are used). A non-default elementary face is a face of which the geometrical surface shape is described by a well ordered two dimensional array of physical (control) points. Note that the geometrical representation of a non-default elementary face is the same as for the input configuration surfaces. The topology is again used to define the ordering of the control points: relation (2) is used for faces. Bicubic Hermite interpolation between the control points is used to describe the surface continuously. A correction procedure is automatically performed such that the surface matches the four face-edge curves exactly. The control points of the non-default elementary faces in a particular multi-block system are stored on the geometry file.

2.3 Interactive domain decomposition

The purpose of the domain modeller is to give users the capability to produce, inspect and modify the topology and geometry file. Hence, the topology and geometry files are input and/or output files of the domain modeller.

The domain modeller allows the content of the topology and geometry file not to be a complete topology, nor a single structure: this gives the users the capability to interrupt the work-session and to restart it at any time. The user can also create temporary dummy entities for reference or to help the building of more complex structures. When the user delivers the topology and geometry files to the grid generator, the not necessary faces, edges and vertices must be removed.

The functions of the domain modeller can be grouped into four sets: the block decomposition functions, the topological object management functions, the input functions and the view manipulation functions.

Conceptually, the only actions performed by the domain modeller are creations, killings and inspections of topological entities. Many different ways to do it are provided by the block decomposition functions.

The user usually starts with a reference geometry, which may be delivered as a set of input configuration surfaces and which is converted into a set of vertices, edges and faces by the domain modeller. The reference geometry can be manipulated in several ways. The user can take the input faces as final block-faces, or he can reface them by creating the block faces as close as possible to the reference geometry, and then projecting them onto

it.

During a typical block decomposition session, vertices and non-default edges and faces are created (or copied from external files). A new block is created by indicating (by mouse) eight block-vertices. The required edges and faces of the new block are then automatically detected and, when they do not already exist, created by default procedures. Usually only a few edges and faces have to be created explicitly; the vast majority is produced by default procedures. Some specific functions help to create new entities by geometrical manipulation of existing ones.

The user can read several sets of topology and geometry files during a work-session: the resulting topology will be the union of the contents of each set. Separate sets of entities can be connected by replacing entities of one set with entities of the other.

Because a complete topology may consist of hundreds of blocks, and it is pratically impossible to handle more than 10 to 20 blocks at once, it is necessary to give the users the capability to operate with sub-sets of topological entities within the domain modeler. Those sub-sets are called "topological objects". A topological object is a topologically consistent set of entities: it means that each entity contained in a topological object is defined by entities which belong to the same object. For example, if an edge is present, also its two vertices must be there.

With the topological object management functions the user can copy and delete entities into and from an object, and he can move entities from an object to another one. When new entities are created, they are automatically copied into a pre-defined object (the so-called "default object"). The topological objects can be set visible or not visible on the graphical screen. The user can select the default object, can assign a name to an object, can list the contents of an object on the screen, and can visualise, with different interpolation modes, the edges and the faces contained in an object. Since the content of each single topological object is a consistent topology, it is possible to write and read the topology and geometry file related to one single object only.

The concept of a topological object as a set of entities is meaningful even if no graphic screen is available. The topological objects are also used by some functions as input sets of entities.

To execute a block-decomposition or a topological object management function, it is necessary to prepare a set of input data. This set contains a code number, which identifies the command to be executed, and usually some other values, depending on the command itself. In order to provide a standard way to input data, whatever command has to be executed, a set of so-called "input functions" was designed. Those functions represent the interface between the user (the keyboard) and domain modeller. It is possible to check the input data before the command is actually executed, and to correct any value, when it is wrong, without re-entering the complete set of data. The command code itself can be changed, without re-entering the other data, when they are correct.

The input functions give the user a complete freedom to choose the sequence of input. The sequence of input functions selected by the user is listed in a so called history file. The history file records a complete block decomposition session and it may therefore be used to replay the block decomposition.

Finally, the view manipulation functions give the capability to visualize the entities on the graphic screen by rotating, translating, zooming, etc.

More details about domain decomposition may be found in [4].

3 Grid generation

The topology and geometry files are input files for the grid generator. Two other files are also important during grid generation: – a grid dimension file which contains the information for the specification of the grid dimensions of the multi-block grid, and – a grid control file which contains the grid control parameters for tuning of the grid. These two files are input and/or output files of the interactive grid generator.

A batch version of the grid generator is also available. The batch version is operational on a supercomputer (NEC-SX3) and is especially useful to create fine grids. The general way of working is as follows. Coarse or medium grids are generated during an interactive grid generation session. The generated grids are defined according to the user specified grid dimensions and grid control parameters. When the user is satisfied about the grid quality of the created grids, then the grid dimensions and grid control parameters are written to the grid dimension file and grid control file which are then output files of the interactive grid generator. Next, the grid dimensions are enlarged by a constant factor (the user has to modify, by an editor, only one number on the grid dimension file), and the complete set of four input files (topology file, geometry file, grid dimension file and grid control file) is sent to the super computer where a fine grid is generated by the batch version of the grid generator. This way of working is successful because of the fact that all grid control parameters have a relative meaning with respect to the grid dimensions.

In subsection 3.1 it is described how the grid dimensions of a particular multi-block system are easily defined. The next three subsections describe the main grid generation procedures for, respectively, edges, faces and blocks. Local grid refinement is described in subsection 3.5. The general way of working during interactive grid generation is described in subsection 3.6.

3.1 Grid dimension specification

The specification of the grid dimensions; i.e. the number of grid cells, of all blocks, faces and edges in a particular multi-block system requires the grid dimension specification of only a few suitable chosen edges. This is due to the constraining effect of the requirement that each grid line in each block must be continuous over any face that the block has in common with any adjacent block. These constraint relations depend only on the topology: each two opposite edges in a face must have the same grid dimension, and each four opposite edges in a block must have the same grid dimension. This observation makes it possible to subdivide the set of edges $\{E\}$ into disjunct sets (called groups) with the property that the grid dimension of all edges in the same group must be the same, while the grid dimensions of two edges in different groups are generally different. Furthermore,

simple sum relations between the grid dimensions of different groups may exist due to the existence of compound edges.If, for instance, a compound edge E with subedges E_1, E_2 belongs to the groups G and G_1, G_2, respectively, then it is clear that the grid dimension of group G is equal to the sum of the grid dimensions of groups G_1 and G_2.

The groups and the sum relations between the groups are automatically generated from the topology. Suppose that a particular multi-block system contains N groups with M sum relations between the groups. Then there are only $N - M$ independent grid dimensions, and the user has to specify the grid dimensions of only $N - M$ suitable chosen edges in order to define the grid dimensions of all groups, and consequently, of all edges, faces and blocks.

3.2 Grid generation in elementary edges

In subsection 2.2 it is described how a smooth curve is constructed for each elementary edge. Such a smooth curve is parameterized according to

$$P_E : s \in [0,1] \mapsto (x,y,z) \in \mathcal{R}^3, \tag{6}$$

where s is the scaled arc length. The orientation is defined by the topology: s runs from vertex V_1 to vertex V_2. The function P_E is called the geometrical shape function of an edge. A grid control function G_E of the form:

$$G_E : \xi \in [0,1] \mapsto s \in [0,1], \tag{7}$$

maps the computational ξ space onto the s space. The orientation of the computational ξ coordinate is the same as for the scaled arc length coordinate s. A grid distribution function maps the computational ξ space onto the edge curve and is defined as the composite mapping $P_E o G_E(\xi) = P_E(G_E(\xi))$. Thus the grid points of an edge with N grid cells are computed according to

$$P_E o G_E(\xi_i),\ \xi_i = i/N,\ i = 0 \ldots N. \tag{8}$$

A grid which is equally distributed along an edge is obtained by taking $s = G_E(\xi) = \xi$. The general form of the function $G_E(\xi)$ is taken as

$$G_E(\xi) = \int_0^{\xi} \exp(\sum_{i=0}^{4} \alpha_i \eta^i) d\eta, \tag{9}$$

where the five coefficients $\alpha_i, i = 0 \ldots 4$ are constants. The chosen form of the function G_E implies that the corresponding stretching function, defined as G_E''/G_E', is a polynomial function. At an elementary edge a user may specify two boundary conditions at each vertex, so that at most four boundary conditions exist. These four boundary conditions, together with the constraint

$$\int_0^1 \exp(\sum_{i=0}^{4} \alpha_i \eta^i) d\eta = 1, \tag{10}$$

are used to compute the five coefficients $\alpha_i, i = 0 \ldots 4$. When the number of boundary conditions is less than four, then the five coefficients are still uniquely determined by demanding that the degree of the polynomial stretching function is as low as possible.

3.3 Grid generation in elementary faces

The surface shape of an elementary face is parameterized by a geometrical shape function of the form:

$$P_F : (s,t) \in [0,1]^2 \mapsto (x,y,z) \in \mathcal{R}^3. \tag{11}$$

The orientation of s and t is defined by the topology: s runs from edge E_1 to E_2 and t runs from edge E_3 to E_4. At the boundary of the unit square in the (s,t) space, the function P_F coincides with the geometrical shape functions of the four face-edges:

$$\begin{aligned} P_F(s,0) = P_{E_3}(s) \quad &, \quad P_F(s,1) = P_{E_4}(s), \\ P_F(0,t) = P_{E_1}(t) \quad &, \quad P_F(1,t) = P_{E_2}(t). \end{aligned} \tag{12}$$

Thus s and t are scaled arc lengths along the boundary of the surface.

Similarly as for elementary edges, a grid control function is used to map the computational (ξ,η) space onto the (s,t) space:

$$G_F : (\xi,\eta) \in [0,1]^2 \mapsto (s,t) \in [0,1]^2. \tag{13}$$

The grid distribution function $P_F o G_F$ maps the computational space onto the surface. The orientation of the (ξ,η) space is the same as for the (s,t) space. The grid points of a face with $N \times M$ grid cells are found by

$$P_F o G_F(\xi_i,\eta_j),\ \xi_i = i/N,\ \eta_j = j/M,\ i = 0 \ldots N,\ j = 0 \ldots M. \tag{14}$$

The grid generation in an elementary face is preceded by the grid generation in the four face-edges. Thus the grid points along the four face-edges are known and also their corresponding s and t values. What remains is the computation of the grid points in the interior of the face.

The computation of the grid control function G_F is equivalent with the computation of the two functions

$$s = s(\xi,\eta)\ ,\ t = t(\xi,\eta). \tag{15}$$

These two functions are known at the boundary of the unit square in the computational domain:

$$\begin{aligned} s(\xi,0) = s_{E_3}(\xi) \quad &, \quad s(0,\eta) = 0, \\ s(\xi,1) = s_{E_4}(\xi) \quad &, \quad s(1,\eta) = 1, \\ t(0,\eta) = t_{E_1}(\eta) \quad &, \quad t(\xi,0) = 0, \\ t(1,\eta) = t_{E_2}(\eta) \quad &, \quad t(\xi,1) = 1. \end{aligned} \tag{16}$$

Note that the functions $s_{E_3}, s_{E_4}, t_{E_1}, t_{E_2}$ are edge grid control functions of the form of Eq. (9) and are thus monotonously increasing.

A simple and robust way to compute (s,t) for values of (ξ,η) in the interior of the unit square is provided by next two equations:

$$s = s_{E_3}(\xi)(1-t) + s_{E_4}(\xi)t, \tag{17}$$

$$t = t_{E_1}(\eta)(1-s) + t_{E_2}(\eta)s. \tag{18}$$

Eq. (17) implies that a grid line ξ = const. is mapped to the unit square in the (s,t) space as a straight line: s is a linear function of t. Eq. (18) implies that a grid line η = const. is also mapped to the unit square in the (s,t) space as a straight line: t is a linear function of s. For given values of ξ and η the corresponding s and t values are found as the intersection point of the two straight lines. It can be easily verified that the grid control function which corresponds to this system has a positive Jacobian i.e. $J = s_\xi t_\eta - s_\eta t_\xi > 0$.

However, the system (17),(18) is completely determined by the grid point distribution along the four face-edges and provides no grid control about the grid line slopes and first grid cell lengths. In order to have more grid control, the system is extended by

$$s = s_{E_3}(\xi)(1-t) + s_{E_4}(\xi)t + f(\xi,t), \tag{19}$$

$$t = t_{E_1}(\eta)(1-s) + t_{E_2}(\eta)s + g(\eta,s). \tag{20}$$

In this grid generation system, Eq. (19) implies that a grid line ξ = const. is mapped to the unit square in the (s,t) space as a curve which can be described as: s is a function of t, and Eq. (20) implies that a grid line η = const. is mapped to the unit square in the (s,t) space as a curve which can be described as: t is a function of s. Again, for given values of ξ and η the corresponding s and t values are found as the intersection point of these two curves.

The functions f and g are correction functions with respect to the straight lines of system (17),(18). Hence, f and g are identical zero at the boundary of the unit squares in respectively the (ξ,t) and (η,s) space. The normal derivatives of f and g at the boundary may be used to define the grid line slopes and first grid cell lengths.

The values of $\partial f/\partial t(\xi,0)$ and $\partial f/\partial t(\xi,1)$ may be used to define the grid line slopes of the grid lines ξ = const. along the face-edges E_3 and E_4, respectively. The values of $\partial f/\partial \xi(0,t)$ and $\partial f/\partial \xi(1,t)$ may be used to control the first grid cell lengths of the grid lines η = const. along the face-edges E_1 and E_2. Similar remarks can be made for the derivatives of the function g. More details about the relation between the normal derivatives of f and g at the boundary of the unit squares and the corresponding grid line slopes and first grid cell lengths at the boundary of the face in physical space may be found in [5].

When the normal derivatives of f and g are defined, then it remains to compute f and g in the interior of the unit squares. This is done by solving the biharmonic equations [6]:

$$\triangle\triangle f = 0, \tag{21}$$

on the unit square $(\xi,t) \in [0,1]^2$, and

$$\triangle\triangle g = 0, \tag{22}$$

on the unit square $(\eta,s) \in [0,1]^2$.

The user control about the grid line slopes is especially useful at singularities. An example is shown in Fig. 2. Such O-type meshes are needed at trailing edges of airfoils, wings, exhaust outlets, etc.

For both grid generation systems (17),(18) and (19),(20), the two families of grid lines in the (s,t) space are determined independently and a grid point is found as the intersection

point of two individual grid lines of each family.This approach is especially successful for Navier-Stokes grids (boundary layers) where the characteristics of the two families of the grid lines are totally different.

3.4 Grid generation in blocks

The grid generation in a block is preceded by grid generation in the six block-faces. Thus, the grid point distribution on the six block faces is known, and what is left to be done is the computation of the grid in the interior of the block. Two methods are used to compute the interior grid points in a block.

The first method is based on trilinear transfinite interpolation in the computational (ξ, η, ζ) space. Thus linear blending functions are used: $\xi, (1-\xi), \eta, (1-\eta), \zeta, (1-\zeta)$.

The second method is an extension of grid generation system (17),(18) for faces. A geometrical shape function is constructed which defines the volume shape of a block:

$$P_B : (s,t,u) \in [0,1]^3 \mapsto (x,y,z) \in \mathcal{R}^3. \tag{23}$$

where the orientation of (s,t,u) is again defined by the the topology. At the boundary of the unit cube in (s,t,u) space, the function P_B coincides with the geometrical shape functions of the six block faces. In fact, the function P_B is constructed by trilinear transfinite interpolation in (s,t,u) space.

A mapping from computational (ξ, η, ζ) onto the (s,t,u) is defined as

$$\begin{aligned}
s &= s_{E_1}(\xi)(1-t)(1-u) + s_{E_2}(\xi)t(1-u) + s_{E_3}(\xi)(1-t)u + s_{E_4}(\xi)tu, && (24)\\
t &= t_{E_5}(\eta)(1-s)(1-u) + t_{E_6}(\eta)s(1-u) + t_{E_7}(\eta)(1-s)u + t_{E_8}(\eta)su, && (25)\\
u &= u_{E_9}(\zeta)(1-s)(1-t) + u_{E_{10}}(\zeta)s(1-t) + u_{E_{11}}(\zeta)(1-s)t + u_{E_{12}}(\zeta)st, && (26)
\end{aligned}$$

where the edges (E_1, E_2, E_3, E_4),(E_5, E_6, E_7, E_8), and $(E_9, E_{10}, E_{11}, E_{12})$ are the sets of four block-edges in respectively the $s-$,$t-$ and $u-$ direction. The functions $s_{E_1}, \ldots, u_{E_{12}}$ are the corresponding edge grid control functions. For given values of ξ,η and ζ, the corresponding s,t and u values are computed by solving Eqs. (24),(25),(26) simultaneously. It can be easily verified that the grid control function which corresponds to this system has a positive Jacobian. Finally, the grid points in the interior of a block are computed by the grid distribution function $P_B o G_B$.

Both methods usually provide a good degree of clustering throughout the grid, but local regions of crossed grid lines, corresponding to negative values of cell volumes sometimes result. However, it appears that negative cell volumes occur much more seldomly when the second method is applied (then grid folding is caused by the geometrical shape function P_B).

3.5 Local Grid refinement

A multi-block grid with the property that each grid line in a block is continuous over any face that the block has in common with any adjacent block is called a basic multi-block

grid. The grid dimensions of a basic multi-block grid are defined in a way as described in subsection 3.1.

However, it is very useful that the grid in a particular block can be refined (coarsened) without changing the grid in the surrounding blocks, so that refined (coarsened) grids can be used in blocks located in regions where large (small) flow gradients are expected.

The grid refinement (coarsening) in a particular block is user-specified by three grid refinement/coarsening factors in each computational direction of that block. Of course, if grid coarsening is applied in a certain computational direction then the grid dimension in that direction must be dividable by the corresponding coarsening factor.

There is one restriction about the way local grid refinement is applied. At a particular internal block-face, there are in general two different grids which belong to the two blocks which have this block-face in common. The restriction is that one of the two grids in the block-face is coarse with respect to the other, so that a grid cell in the coarse grid may be connected to a number of fine grid cells in the fine grid. This property facilitates a flow solver to remain conservative across block-faces: the flux through a coarse grid cell-face is given by the sum of the fluxes through the corresponding fine-grid cell-faces [7]. An example of the application of local grid refinement is shown in Fig. 3.

3.6 Interactive grid generation

The interactive grid generator is operational on the Silicon Graphics Iris 4D workstation family.

During an interactive session, the first action of a user is to read the topology and geometry file of a particular multi-block system. After that, a selection panel is available to visualize the topology and geometry. The panel contains all block labels and the user may select (by mouse) a number of blocks from the panel. The result is a screen which depicts the selected blocks. The topology of each block is represented by the six block-face labels, all elementary and compound face labels of which a block-face may consist of, all corresponding elementary and compound edge labels, and all vertex labels (the total number of vertices is in general more than eight). The geometry of each block is represented only by the geometrical shape of all elementary edges on the block boundary (the total number of elementary edges on the block boundary is more than twelve in general).

Next, the grid dimensions are specified. The mouse is used to select an edge from the screen, and a number is given (by keyboard) which defines the number of grid cells along the edge. With the group concept described in subsection 3.1, the program automatically identifies those edges which obtain the same grid dimension value. If there are compound edges, then the sum relations between the groups are used to check if it is possible to compute the dimensions of other edges. If an edge dimension is known then the colour of the edge on the screen is changed and the grid dimension value appears at the middle of the edge.

This process is repeated until the dimensions of all edges are known. The user may then write the grid dimension file which defines the grid dimensions of the multi-block system.

If a grid dimension file already exists then the process of specifying the grid dimensions may be skipped and the user can simply read the grid dimension file.

When the grid dimensions are known, a second selection panel becomes available which contains the i, j, k values of all grid planes in each block. If the user selects some i, j or k values of some blocks then the corresponding grid planes in the blocks are shown on the screen. Thus the second selection panel is used to specify which grid planes must be visualized while the first selection panel is used to specify which blocks must be visualized topologically. In the interactive grid generator there are no explicit commands to compute a grid in some edges, faces or blocks, but instead of that, the visualization of grid planes in blocks is automatically converted to commands to compute the grids in the corresponding block-faces (if a grid plane in a block corresponds with a block-face) or blocks (if a grid plane is an interior grid plane).

Next, the grid tuning process may start. Along elementary edges, grid control is available only at the two vertices of the edges. The user may specify a grid density ρ at a vertex of an elementary edge by selecting the edge and vertex (via their labels) from the screen by mouse and by defining ρ by keyboard. The result is that the first grid cell length along the edge at the vertex becomes $\rho L/N$ where L is the lenth of the edge and N is the number of grid cells along the edge. It is also possible to specify a stretching value R at a vertex of an elementary edge. The result is that the ratio between the second and first grid cell length at the vertex becomes $1 + (R/N)$. Finally, the user may "connect" edges. If an edge E_2 at vertex V_2 is connected to an edge E_1 at vertex V_1 then the first grid cell length along edge E_2 at V_2 becomes equal to the first grid cell length along edge E_1 at V_1. In this way large chains of connected edges may be constructed, and if the grid in the "mother" edge is changed, the grid in all other edges in the chain are then also automatically changed. The program automatically takes care that a chain is not closed. The connection of edges is very useful to construct smooth grids across elementary edges.

Grid generation in an elementary face is preceded by grid generation in the four face-edges (the program automatically computes the grid in the four face-edges before it computes the grid in the interior of a face). Thus the grid points along the four face-edges are known. Grid generation in elementary faces may be based on system (17),(18) or system (19),(20). If system (17),(18) is used then the grid in an elementary face is determined by the grid point distribution along the four face-edges only. If system (19),(20) is used, then the grid in an elementary face is also determined by the grid line slopes and first grid cell lengths along the four face edges. At the corners of the face, the angle between the two face-edges is known and also the first grid cell lengths along the two face-edges. Default, these angles and grid cell lengths at the corners are interpolated (using arclengths) to define the grid line slopes and first grid cell lengths on the four face edges. Then the boundary conditions are sufficient to solve system (19),(20). However, the user may also specify explicitly the grid line slope and first grid cell length at certain locations along the four face-edges (which is for instance necessary at singularities). In that case, the mouse is used to identify the location along a face-edge, and the grid line slope angle and first grid cell length are specified by keyboard.

Grid generation in blocks is preceded by grid generation in the six block-faces (the

program automatically computes the grid in the six block-faces before it computes the grid in the interior of the block). Thus the grid points on the six block-faces are known. Grid generation in a block may be based on trilinear transfinite interpolation in computational or arclength scaled space. For both methods the grid in the block is determined by the grid point distribution on the six block-faces only. However, grid folding occurs much more seldomly when trilinear transfinite interpolation in arclength scaled space is applied.

During the interactive session the user may change at any moment the grid dimensions or apply local grid refinement. Local grid refinement in a block is performed by specifying the grid refinement/coarsening factors in the three computational directions of a block. It is not necessary to change the values of the grid tuning parameters when the grid dimensions are changed because their values are relative with respect to the grid dimensions.

One interactive session is in general not sufficient to tune the complete grid for a complex configuration. Therefore at the end of a session the user may write the grid dimension and grid control file (which contains all grid tuning parameters) and also a grid file which contains the partially constructed grid. Then, in a next session, the user can read these files and continue the grid generation process.

There are of course more additional "tools" in the interactive grid generator. For example, for grid inspection it is necessary to zoom, rotate and translate. Furthermore, it is possible at any moment to switch on and of the labels of the blocks, faces, edges and vertices which are selected via the first selection panel.

4 Applications

The usability of the applied domain decomposition and grid generation techniques is demonstrated for both aerodynamic and hydrodynamic configurations.

Complex aerodynamic configurations are shown in Figs. 4 and 5. Some results about the numerical flow simulation about the Fokker 50 and Fokker 100 may be found in [8]. Fig. 5a shows the aerodynamic surface of the Alenia transport aircraft G222 and Fig. 5b shows the corresponding grid.

Hydrodynamic configurations are shown in Figs. 6 and 7. Fig. 6 shows the grid in a part of the river Rhine. High grid density is not required at the boundary but in the interior of the river due to a minor bed which is a typical hydraulic feature. The topology contains five blocks. Fig. 7 is an example of a complex hydraulic structure to be built in the river Maas. The resulting topology is shown in Fig. 7a, the surface grid is shown in Fig. 7b. Note that local grid refinement has been applied in one block. Fig. 7c shows the grid in an interior circular plane. Note that such a plane contains four O-type singularities. More details about block decomposition and grid generation for hydrodynamic configurations may be found in [9].

Finally, Fig. 8 illustrates that it is possible to generate Navier-Stokes grids with the existing grid generation techniques.

5 References

[1] J.W. Boerstoel, J.M.J.W. Jacobs, A. Kassies, A. Amendola, R. Tognaccini, P.L. Vitagliano, Design and testing of a multiblock grid generation procedure for aircraft design and research. Applications of Mesh Generation to Complex 3-D Configurations, AGARD-CP-464,1990.

[2] L.E. Eriksson, E. Orbekk, Algebraic Block-Structured Grid Generation Based on a Macro-Block Concept. Applications of Mesh Generation to Complex 3-D Configurations, AGARD-CP-464,1990.

[3] J.W. Boerstoel, S.P. Spekreijse, An Information System for the Numerical Simulation of 3D Euler Flows around Aircraft. Computer Methods in Applied Mechanics and Engineering 89, pp. 237-257, 1991.

[4] P.L. Vitagliano, J.W. Boerstoel, Introduction and Guide to the Use of EDOMO for the Graphical Interactive Decomposition of Flow Domains into Blocks. NLR CR 90265, 1991.

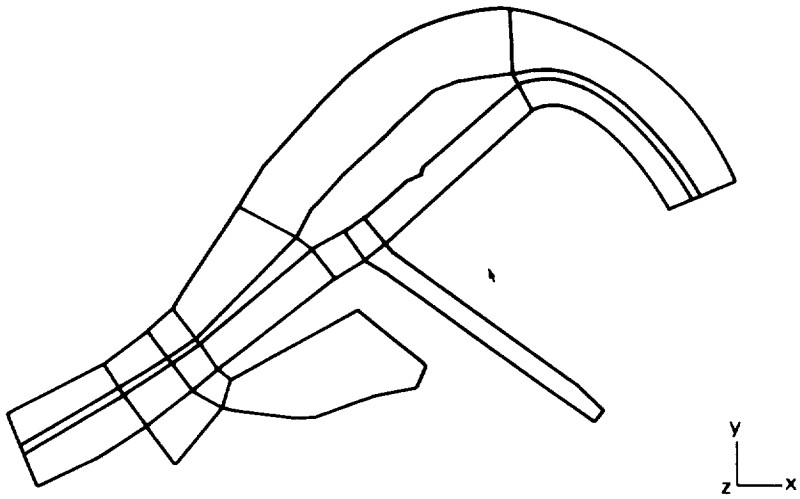
[5] S.P. Spekreijse, J.W. Boerstoel, New Concepts for Multi-Block Grid Generation for Flow Domains Around Complex Aerodynamic Configurations. Numerical Grid Generation in Computational Fluid Dynamics and Related Fields, ed. A.S. Arcilla e.a., Elsevier North-Holland,1991.

[6] P.E. Bjorstad, Numerical Solution of the Biharmonic Equation. Ph.D.dissertation, Stanford University, 1980.

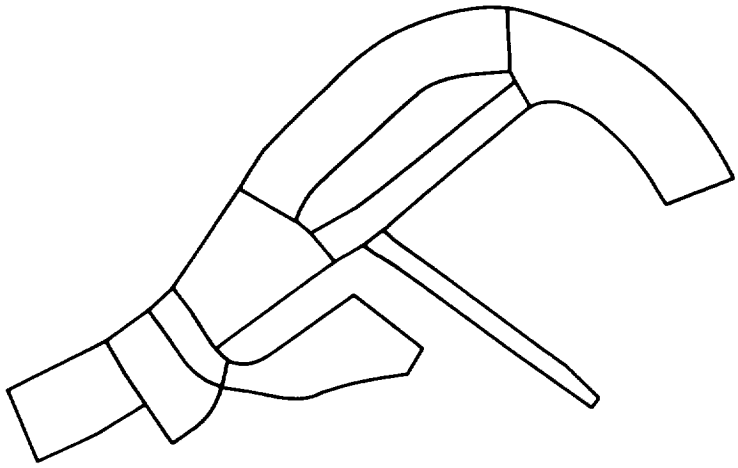
[7] A. Kassies, R. Tognaccini, Boundary Conditions for Euler Equations at Internal Block Faces of Multiblock Domains Using Local Grid Refinement, AIAA Paper 90-1590, 1990.

[8] J.L. Kuyvenhoven, J.W. Boerstoel, 3D Euler Flows Around Modern Airplanes. Proc. Eight GAMM-Conference on Numerical Methods in Fluid Dynamics, Notes on Numerical Fluid Mechanics, 29, ed. P. Wesseling. Vieweg 1989.

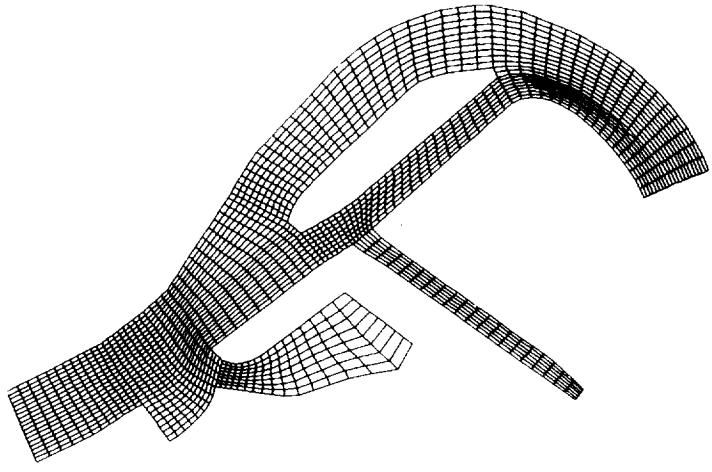
[9] M.J. van der Marel, Evaluation of ENGRID and EDOMO. Delft Hydraulics report Q997,1991.



a) Complete block boundary interfacing



b) Partial block boundary interfacing



c) Grid

Fig. 1 Part of harbour of Rotterdam near Noordereiland

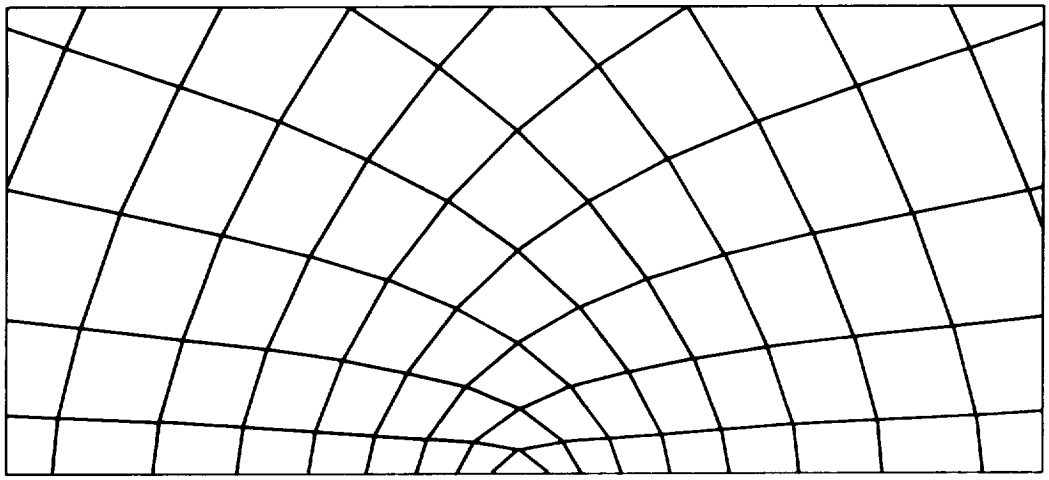


Fig. 2 Grid near O-type singularity

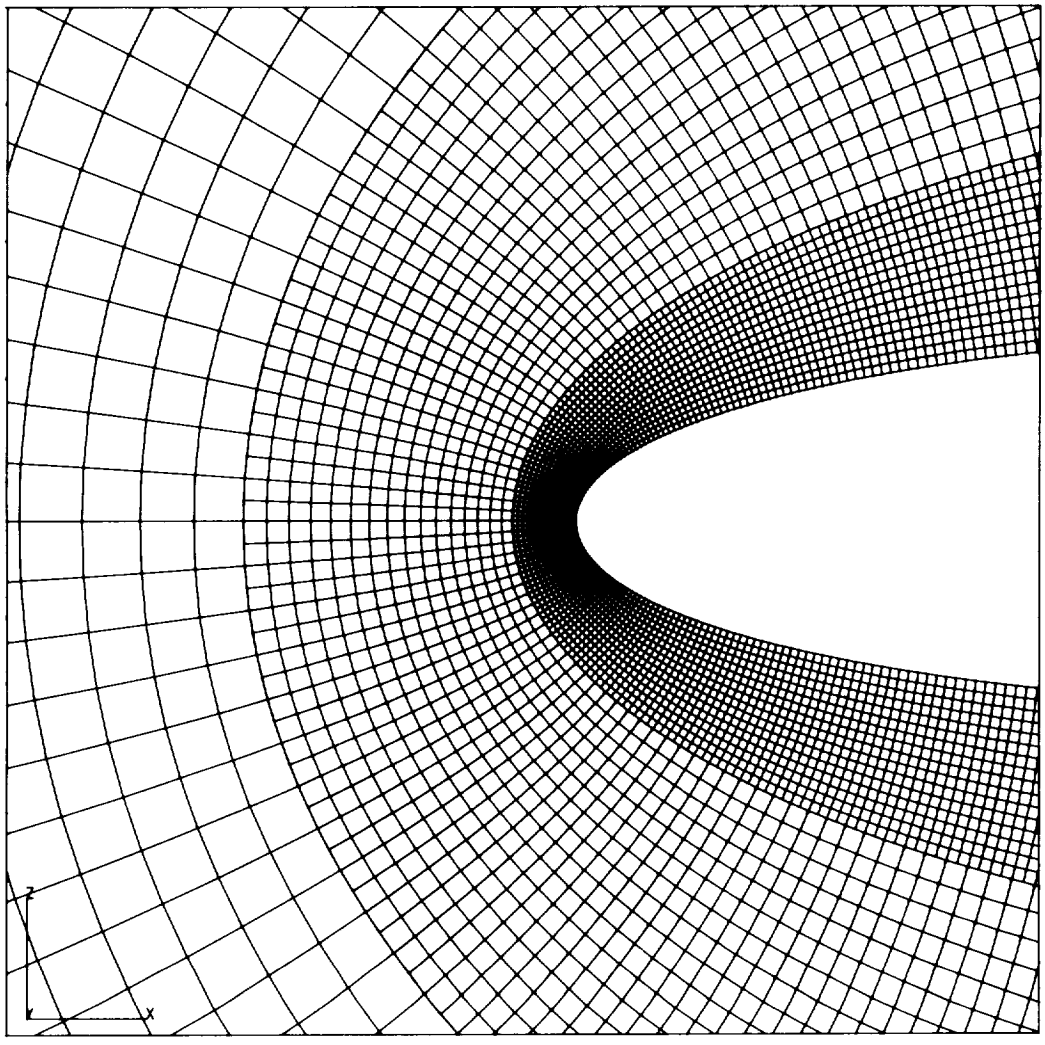


Fig. 3 Leading edge region of 3-block fine grid

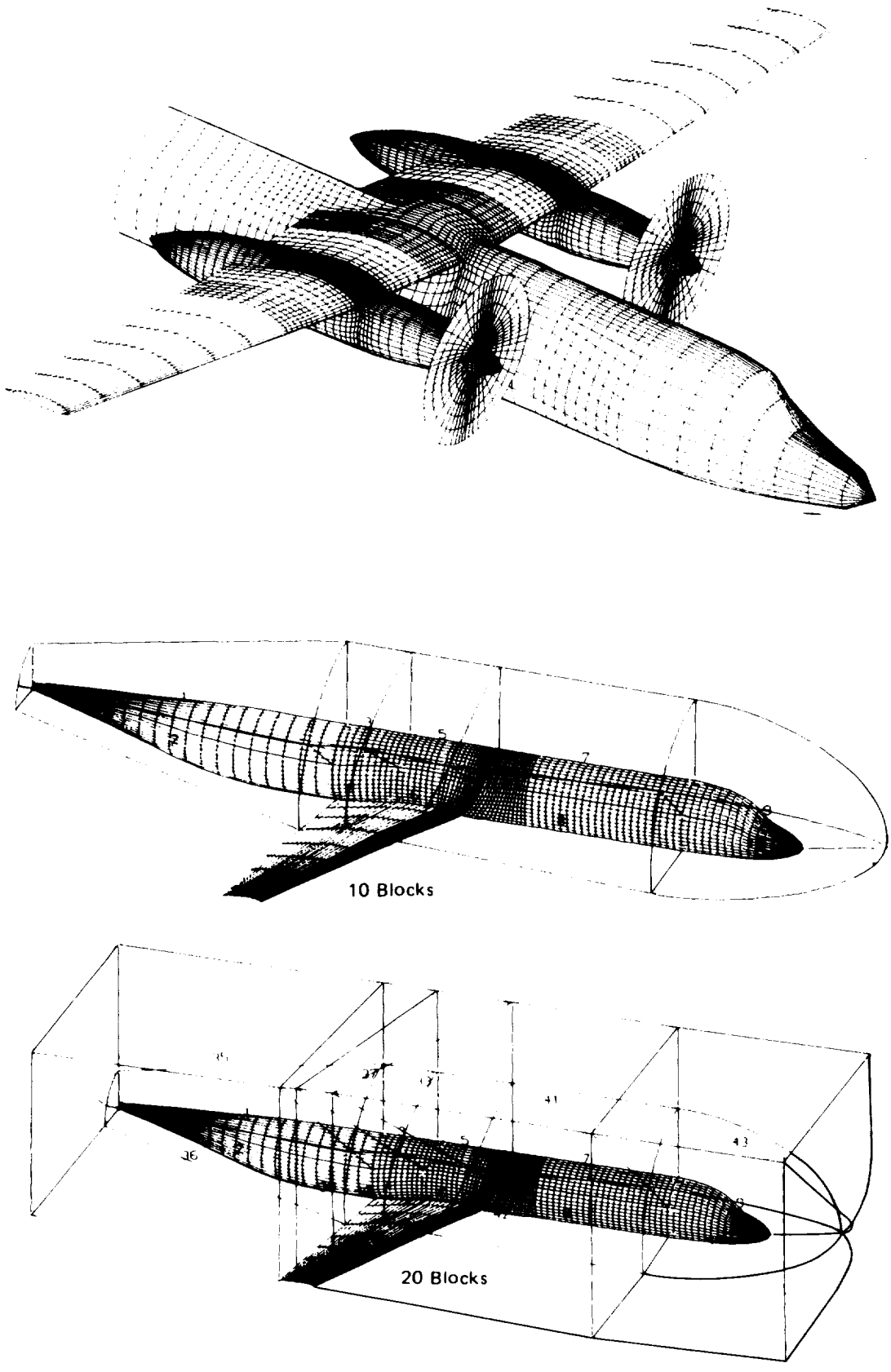


Fig. 4 Fokker 50 and Fokker 100 configurations

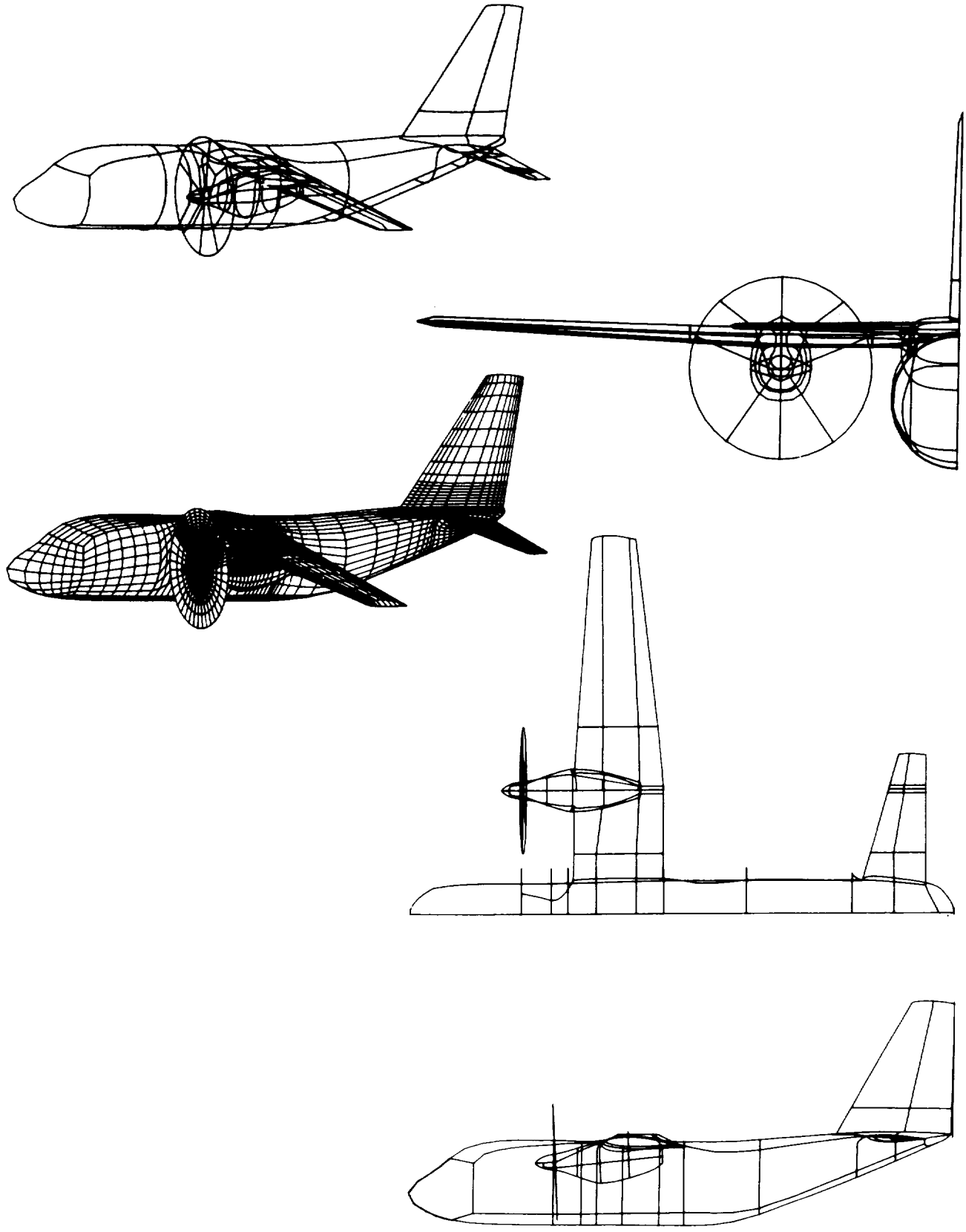


Fig. 5a Aerodynamic surface of Alenia transport aircraft G222

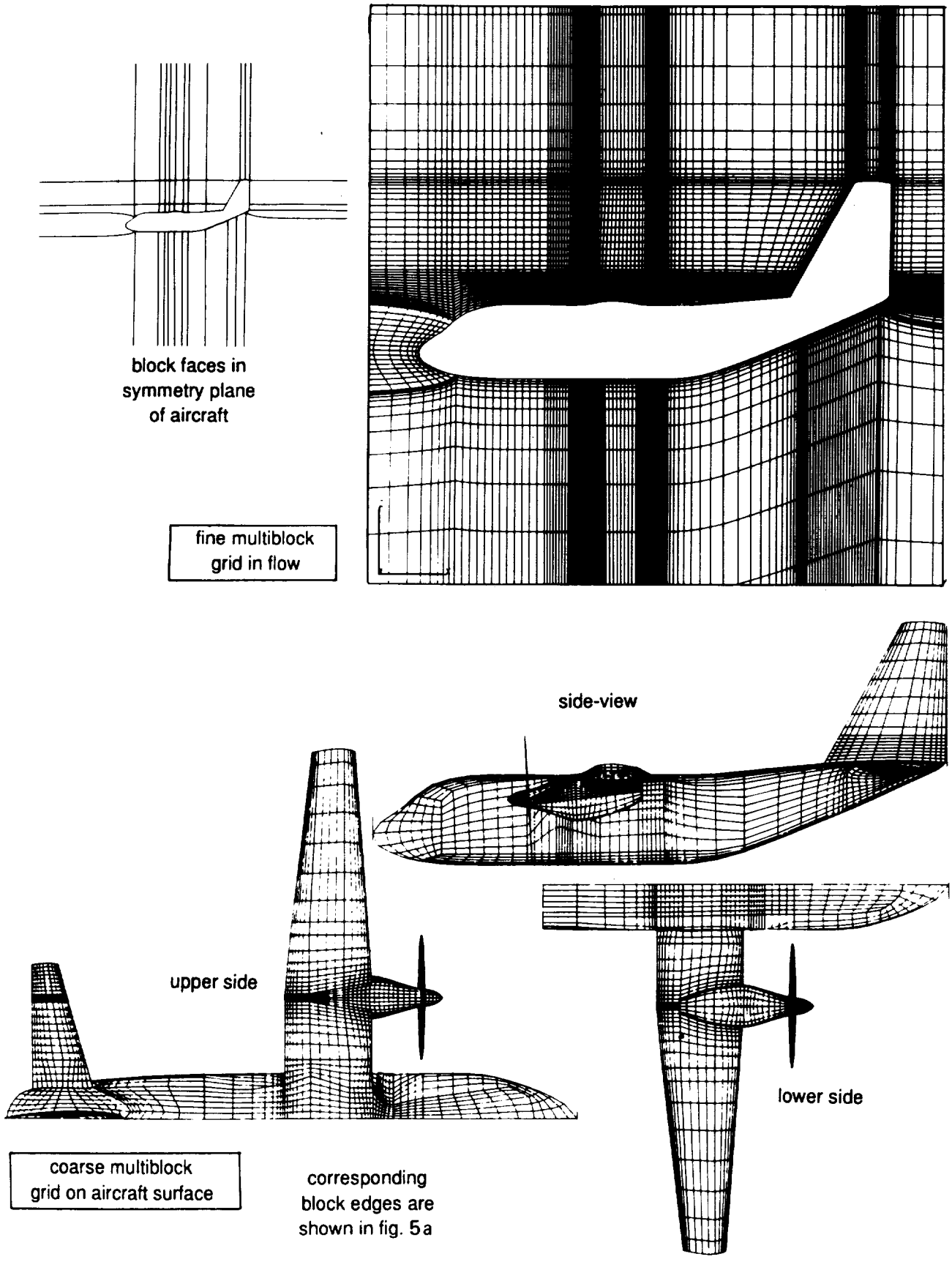


Fig. 5b Corresponding multi-block grid on aircraft surface and symmetry plane

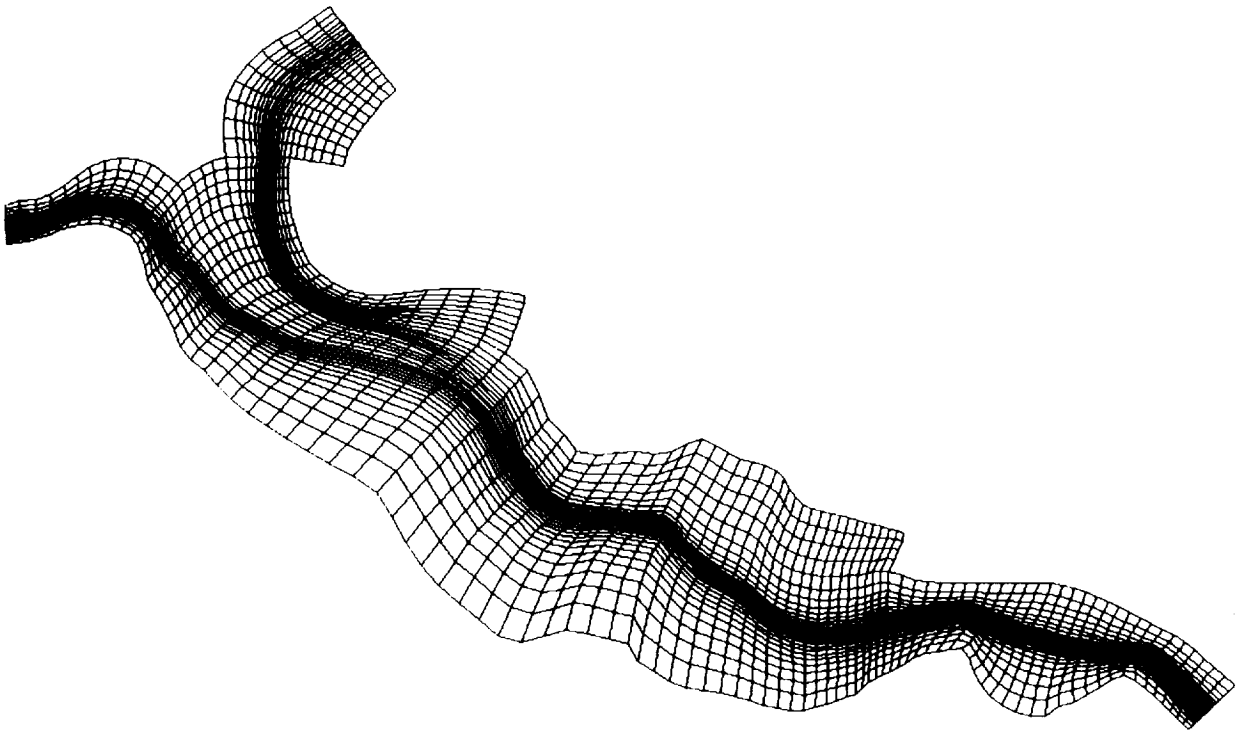
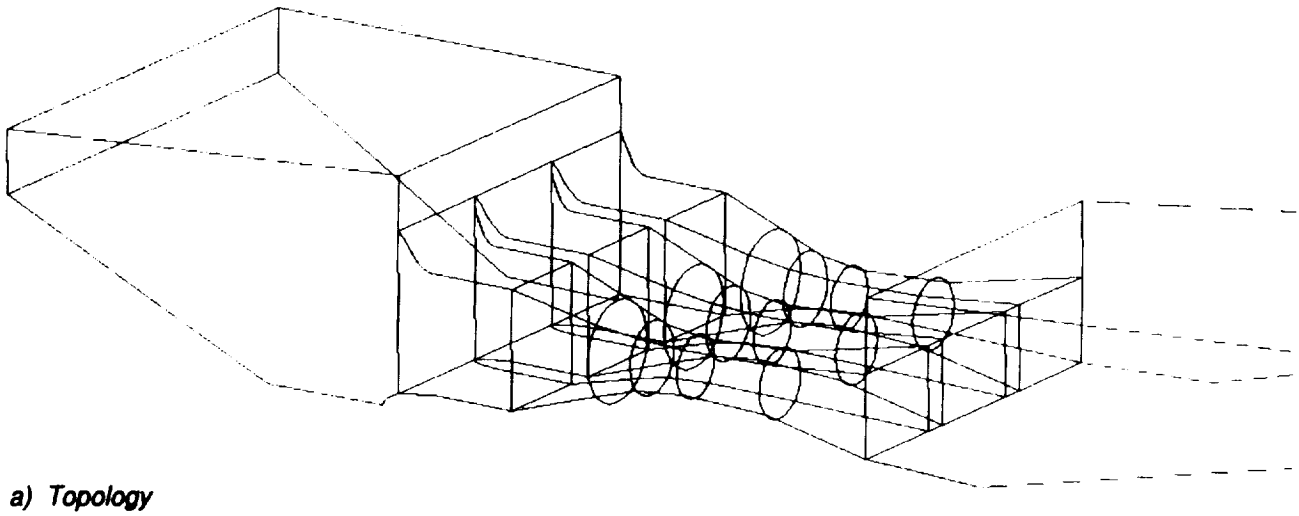
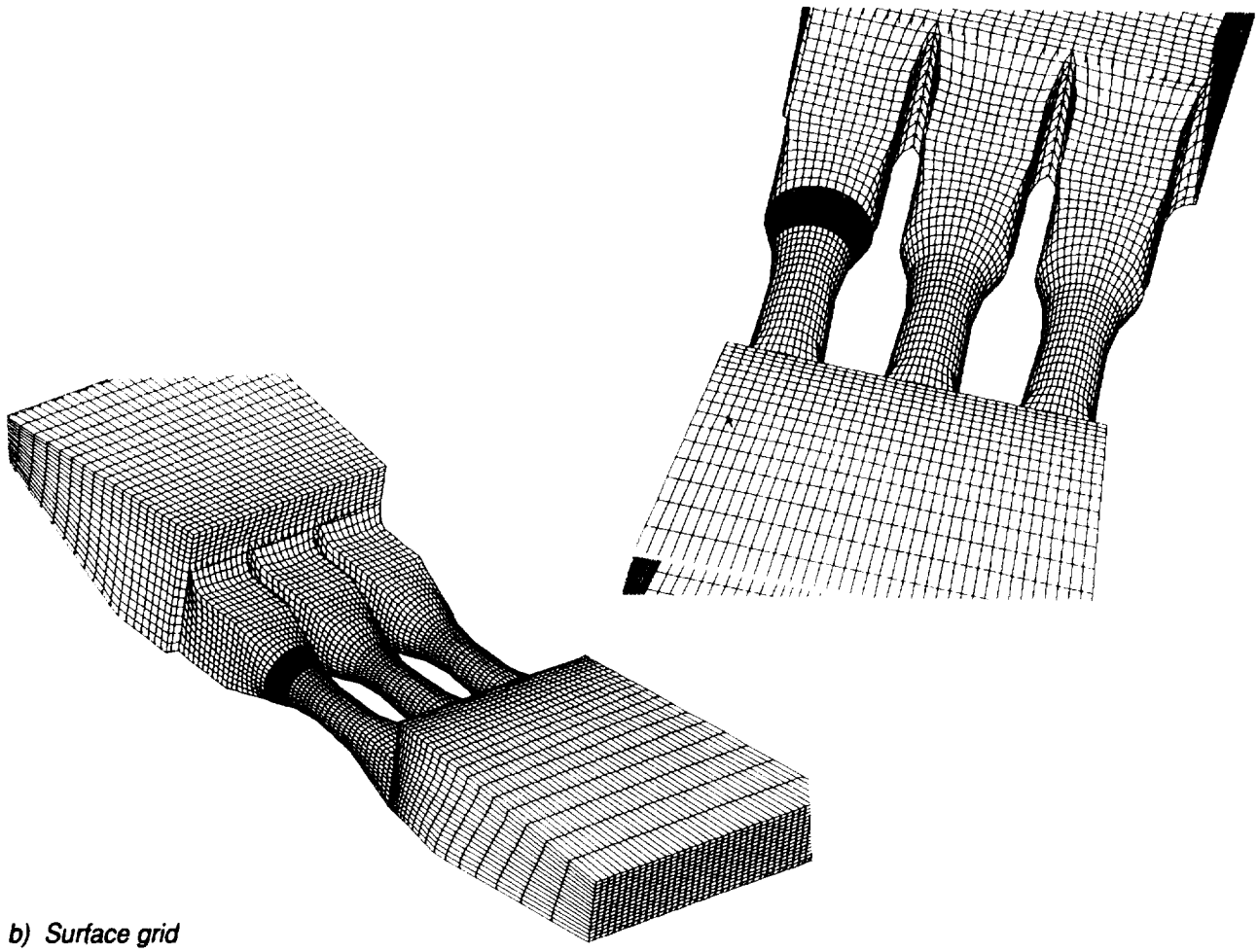


Fig. 6 Grid in part of the river Rhine

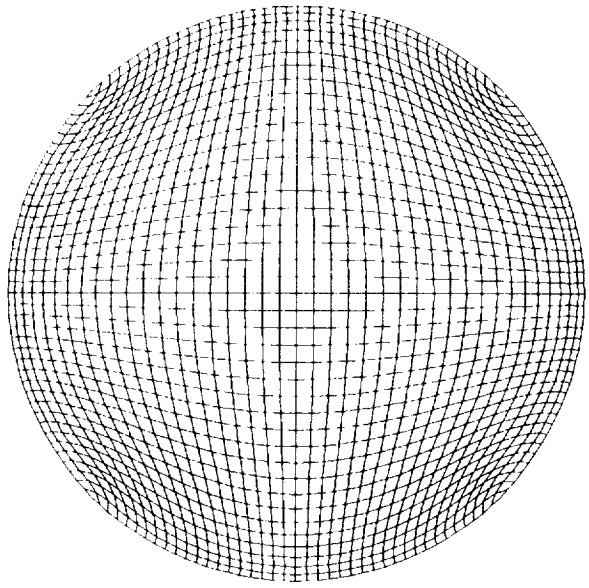


a) Topology

Fig. 7 Water power station



b) Surface grid



c) Grid in interior circular plane

Fig. 7 Water power station (continued)

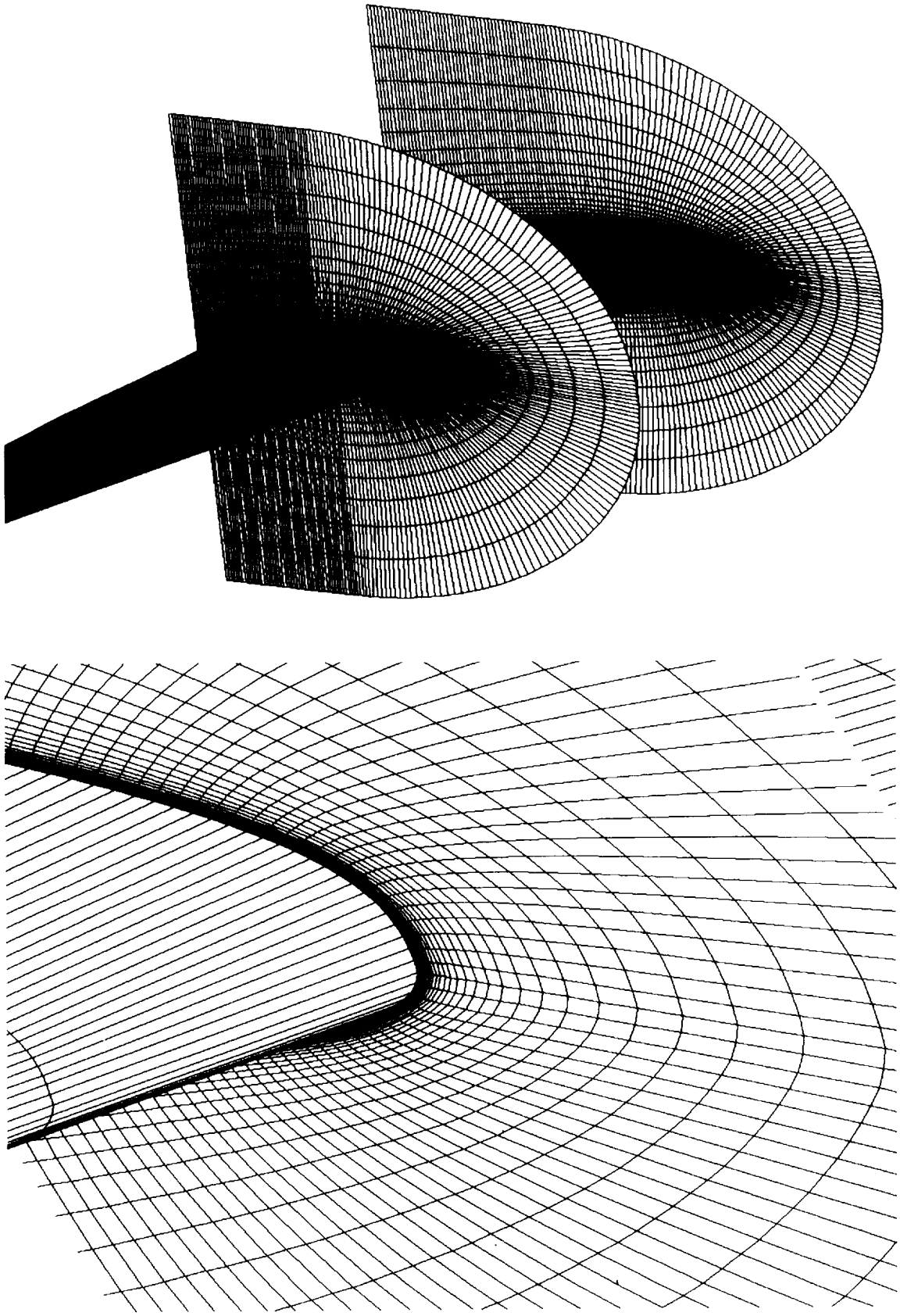


Fig. 8 Navier-Stokes grid around a wing